Overview of Assembler for Project 1, CISC 340

By Abenezer Ayana and Austin Hippe

This project was focused on creating a C program that translates an assembly-language instructions into machine code. To accomplish this, we created a separate function called createMachineCode() that takes a filename pointer and an integer array as input and calculates the machine code. We also created a stringCompare() function that compares a string input to any of the 8 UST-3400 instructions and ".fill" (stored as a defined array at the top of our code), which helps us to decide if a token is an instruction or a label. Our main function utilizes GNU getopt to access the user-inputted assembly file. It uses the line count in the file to create an array that will ultimately store the machine code. It then sends this array and the file pointer optarg to the createMachineCode() function. We then make three passes through the file to decide the memory address values, number of symbolic labels, and maximum label length. These values help us to create two separate arrays called symbolic_labels and symbolic_values, which store the labels and memory address of the labels respectively. We perform this task carefully, each time checking if the requirements for a label are fulfilled (maximum length of six, no special characters, etc.) using our isCorrectLabel() function. Once these arrays are populated, we begin our instruction encoding line by line. After making the necessary checks to determine the label, the opcode, and the field amount, we use left shift (<<) to encode our instruction. Since our opcode field spans from bits 24-22, we shift our opcode 22 bits to the left onto an int variable. We then decide the next shifts using the type of our opcode (Our opcodes are stored in an array so that we can search for making this distinction). For R-types (add,nand), we shift the last two register field values by 19 and 16 bits to the left respectively and do Bitwise Or (|) for the first register field value. For I-types (lw,sw,beq) we shift our first two register field values by 19 and 16 bits to the left respectively. Afterwards, we check if our offset field is a label or a value. This depends on the type of offset we find (label or numeric). For labels, we compute the difference of our label value (found using our symbolic_values array) and PC+1; for a numeric offset we simply convert the value to integers using atoi function. Because this field needs to be 16 bits, we mask out the highest 16 bits using Bitwise AND operator and 0x0000FFFF value, and finally combine it with our result using Bitwise OR. For J-type (jalr), we simply shift our two register values by 19 and 16 bits to the left respectively. Finally, we don't perform any shifts for O-types (noop,halt). We then store each of these results into an array, which uses line count as a size. After this process is concluded, we revisit the array passed by main and copy all machine codes to it from our result array using our copyArrayContent() function. If getopt receives one input, we print this array using our printArray() function. If our getopt receives output value, we write to the output file using our writeArray() function. We also implemented additional functions such as trim() and checkDuplicate() to help us with token processing and error checking. The difficulty we faced during the implementation of our code was applying the offset mask using Bitwise And, as we were previously using a 16-bit type int to store the result from the mask. Currently, our

program functions correctly and prints the accurate machine code or error message depending on the getopt inputs.